Monotonic Infinite Lookback Attention for Simultaneous Machine Translation

Naveen Arivazhagan* **Colin Cherry*** **Wolfgang Macherey** **Chung-Cheng Chiu**

Semih Yavuz **Ruoming Pang** **Wei Li** **Colin Raffel**

Google

navari,colincherry,wmach,chungchengc@google.com

syavuz,rpang,mweili,craffel@google.com

Abstract

Simultaneous machine translation begins to translate each source sentence before the source speaker is finished speaking, with applications to live and streaming scenarios. Simultaneous systems must carefully schedule their reading of the source sentence to balance quality against latency. We present the first simultaneous translation system to learn an adaptive schedule jointly with a neural machine translation (NMT) model that attends over all source tokens read thus far. We do so by introducing Monotonic Infinite Lookback (MILk) attention, which maintains both a hard, monotonic attention head to schedule the reading of the source sentence, and a soft attention head that extends from the monotonic head back to the beginning of the source. We show that MILk's adaptive schedule allows it to arrive at latency-quality trade-offs that are favorable to those of a recently proposed wait-k strategy for many latency values.

1 Introduction

Simultaneous machine translation (MT) addresses the problem of how to begin translating a source sentence before the source speaker has finished speaking. This capability is crucial for live or streaming translation scenarios, such as speech-to-speech translation, where waiting for one speaker to complete their sentence before beginning the translation would introduce an intolerable delay. In these scenarios, the MT engine must balance latency against quality: if it acts before the necessary source content arrives, translation quality degrades; but waiting for too much source content can introduce unnecessary delays. We refer to the strategy an MT engine uses to balance reading source tokens against writing target tokens as its **schedule**.

*Equal contributions.

Recent work in simultaneous machine translation tends to fall into one of two bins:

- The schedule is learned and/or adaptive to the current context, but assumes a fixed MT system trained on complete source sentences, as typified by wait-if-* (Cho and Esipova, 2016) and reinforcement learning approaches (Grissom II et al., 2014; Gu et al., 2017).
- The schedule is simple and fixed and can thus be easily integrated into MT training, as typified by wait-k approaches (Dalvi et al., 2018; Ma et al., 2018).

Neither scenario is optimal. A fixed schedule may introduce too much delay for some sentences, and not enough for others. Meanwhile, a fixed MT system that was trained to expect complete sentences may impose a low ceiling on any adaptive schedule that uses it. Therefore, we propose to train an adaptive schedule jointly with the underlying neural machine translation (NMT) system.

Monotonic attention mechanisms (Raffel et al., 2017; Chiu and Raffel, 2018) are designed for integrated training in streaming scenarios and provide our starting point. They encourage streaming by confining the scope of attention to the most recently read tokens. This restriction, however, may hamper long-distance reorderings that can occur in MT. We develop an approach that removes this limitation while preserving the ability to stream.

We use their hard, monotonic attention head to determine how much of the source sentence is available. Before writing each target token, our learned model advances this head zero or more times based on the current context, with each advancement revealing an additional token of the source sentence. A secondary, soft attention head can then attend to any source words at or before that point, resulting in **M**onotonic **I**nfinite

Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 1313–1323
Florence, Italy, July 28 - August 2, 2019.

Lookbac**k** (MILk) attention. This, however, removes the memory constraint that was encouraging the model to stream. To restore streaming behaviour, we propose to jointly minimize a latency loss. The entire system can efficiently be trained in expectation, as a drop-in replacement for the familiar soft attention.

Our contributions are as follows:

1. We present MILk attention, which allows us to build the first simultaneous MT system to learn an adaptive schedule jointly with an NMT model that attends over all source tokens read thus far.

2. We extend the recently-proposed Average Lagging latency metric (Ma et al., 2018), making it differentiable and calculable in expectation, which allows it to be used as a training objective.

3. We demonstrate favorable trade-offs to those of wait-k strategies at many latency values, and provide evidence that MILk's advantage extends from its ability to adapt based on source content.

2 Background

Much of the earlier work on simultaneous MT took the form of strategies to chunk the source sentence into partial segments that can be translated safely. These segments could be triggered by prosody (Fügen et al., 2007; Bangalore et al., 2012) or lexical cues (Rangarajan Sridhar et al., 2013), or optimized directly for translation quality (Oda et al., 2014). Segmentation decisions are surrogates for the core problem, which is deciding whether enough source content has been read to write the next target word correctly (Grissom II et al., 2014). However, since doing so involves discrete decisions, learning via back-propagation is obstructed. Previous work on simultaneous NMT has thus far side-stepped this problem by making restrictive simplifications, either on the underlying NMT model or on the flexibility of the schedule.

Cho and Esipova (2016) apply heuristics measures to estimate and then threshold the confidence of an NMT model trained on full sentences to adapt it at inference time to the streaming scenario. Several others use reinforcement learning (RL) to develop an agent to predict read and write decisions (Satija and Pineau, 2016; Gu et al., 2017; Alinejad et al., 2018). However, due to computational challenges, they pre-train an NMT model on full sentences and then train an agent that sees the fixed NMT model as part of its environment.

Dalvi et al. (2018) and Ma et al. (2018) use fixed schedules and train their NMT systems accordingly. In particular, Ma et al. (2018) advocate for a wait-k strategy, wherein the system always waits for exactly k tokens before beginning to translate, and then alternates between reading and writing at a constant pre-specified emission rate. Due to the deterministic nature of their schedule, they can easily train the NMT system with the schedule in place. This can allow the NMT system to learn to anticipate missing content using its inherent language modeling capabilities. On the downside, with a fixed schedule the model cannot speed up or slow down appropriately for particular inputs.

Press and Smith (2018) recently developed an attention-free model that aims to reduce computational and memory requirements. They achieve this by maintaining a single running context vector, and eagerly emitting target tokens based on it whenever possible. Their method is adaptive and uses integrated training, but the schedule itself is trained with external supervision provided by word alignments, while ours is latent and learned in service to the MT task.

3 Methods

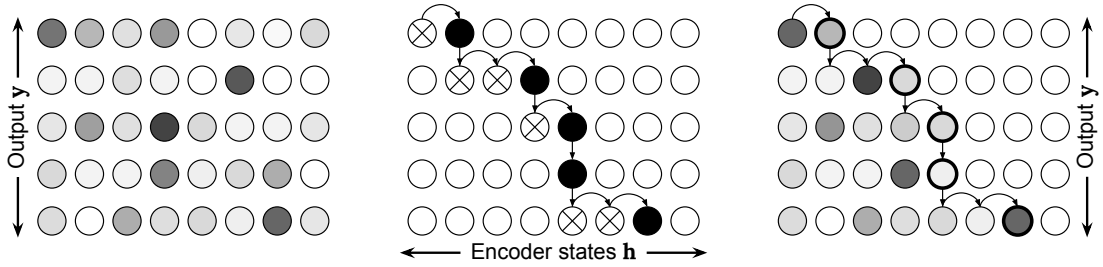
In sequence-to-sequence modeling, the goal is to transform an input sequence $\mathbf{x} = \{x_1, \ldots, x_{|\mathbf{x}|}\}$ into an output sequence $\mathbf{y} = \{y_1, \ldots, y_{|\mathbf{y}|}\}$. A sequence-to-sequence model consists of an encoder which maps the input sequence to a sequence of hidden states and a decoder which conditions on the encoder output and autoregressively produces the output sequence. In this work, we consider sequence-to-sequence models where the encoder and decoder are both recurrent neural networks (RNNs) and are updated as follows:

$$h_j = \text{EncoderRNN}(x_j, h_{j-1}) \quad (1)$$

$$s_i = \text{DecoderRNN}(y_{i-1}, s_{i-1}, c_i) \quad (2)$$

$$y_i = \text{Output}(s_i, c_i) \quad (3)$$

where h_j is the encoder state at input timestep j, s_i is the decoder state at output timestep i, and c_i is a context vector. The context vector is computed based on the encoder hidden states through the use of an attention mechanism (Bahdanau et al.,



(a) Soft attention. (b) Monotonic attention. (c) MILk attention.

Figure 1: Simplified diagrams of the attention mechanisms discussed in Sections 3.1 and 3.2. The shading of each node indicates the amount of attention weight the model assigns to a given encoder state (horizontal axis) at a given output timestep (vertical axis).

2014). The function $\text{Output}(\cdot)$ produces a distribution over output tokens y_i given the current state s_i and context vector c_i. In standard soft attention, the context vector is computed as follows:

$$e_{i,j} = \text{Energy}(h_j, s_{i-1}) \quad (4)$$

$$\alpha_{i,j} = \text{softmax}(e_{i,:})_j := \frac{\exp(e_{i,j})}{\sum_{k=1}^{T} \exp(e_{i,k})} \quad (5)$$

$$c_i = \sum_{j=1}^{|\mathbf{x}|} \alpha_{i,j} h_j \quad (6)$$

where $\text{Energy}()$ is a multi-layer perceptron.

One issue with standard soft attention is that it computes c_i based on the entire input sequence for all output timesteps; this prevents attention from being used in streaming settings since the entire input sequence needs to be ingested before generating any output. To enable streaming, we require a schedule in which the output at timestep i is generated using just the first t_i input tokens, where $1 \leq t_i \leq |\mathbf{x}|$.

3.1 Monotonic Attention

Raffel et al. (2017) proposed a monotonic attention mechanism that modifies standard soft attention to provide such a schedule of interleaved reads and writes, while also integrating training with the rest of the NMT model. Monotonic attention explicitly processes the input sequence in a left-to-right order and makes a hard assignment of c_i to one particular encoder state denoted h_{t_i}. For output timestep i, the mechanism begins scanning the encoder states starting at $j = t_{i-1}$. For each encoder state, it produces a Bernoulli selection probability $p_{i,j}$, which corresponds to the probability of either stopping and setting $t_i = j$, or else moving on to the next input timestep, $j+1$, which represents reading one more source token. This selection probability is computed through the use of an energy function that is passed through a logistic sigmoid to parameterize the Bernoulli random variable:

$$e_{i,j} = \text{MonotonicEnergy}(s_{i-1}, h_j) \quad (7)$$

$$p_{i,j} = \sigma(e_{i,j}) \quad (8)$$

$$z_{i,j} \sim \text{Bernoulli}(p_{i,j}) \quad (9)$$

If $z_{i,j} = 0$, j is incremented and these steps are repeated; if $z_{i,j} = 1$, t_i is set to j and c_i is set to h_{t_i}.

This approach involves sampling a discrete random variable and a hard assignment of $c_i = h_{t_i}$, which precludes backpropagation. Raffel et al. (2017) instead compute the probability that $c_i = h_j$ and use this to compute the expected value of c_i, which can be used as a drop-in replacement for standard soft attention, and which allows for training with backpropagation. The probability that the attention mechanism attends to state h_j at output timestep i is computed as

$$\alpha_{i,j} = p_{i,j} \left((1 - p_{i,j-1}) \frac{\alpha_{i,j-1}}{p_{i,j-1}} + \alpha_{i-1,j} \right) \quad (10)$$

There is a solution to this recurrence relation which allows $\alpha_{i,j}$ to be computed for all j in parallel using cumulative sum and cumulative product operations; see Raffel et al. (2017) for details.

Note that when $p_{i,j}$ is either 0 or 1, the soft and hard approaches are the same. To encourage this, Raffel et al. (2017) use the common approach of adding zero-mean Gaussian noise to the logistic sigmoid function's activations. Equation 8 becomes:

$$p_{i,j} = \sigma\left(e_{i,j} + \mathcal{N}(0, n)\right) \quad (11)$$

One can control the extent to which $p_{i,j}$ is drawn toward discrete values by adjusting the noise variance n. At run time, we forgo sampling in favor of simply setting $z_{i,j} = \mathbb{1}(e_{i,j} > 0)$.

While the monotonic attention mechanism allows for streaming attention, it requires that the decoder attend only to a single encoder state, h_{t_i}. To address this issue, Chiu and Raffel (2018) proposed monotonic chunkwise attention (MoChA), which allows the model to perform soft attention over a small fixed-length chunk preceding t_i, i.e. over all available encoder states, $h_{t_i-cs+1}, h_{t_i-cs+2}, \ldots, h_{t_i}$ for some fixed chunk size cs.

3.2 Monotonic Infinite Lookback Attention

In this work, we take MoChA one step further, by allowing the model to perform soft attention over the encoder states $h_1, h_2, \ldots, h_{t_i}$. This gives the model "infinite lookback" over the past seen thus far, so we dub this technique **M**onotonic **I**nfinite **L**ookbac**k** (MILk) attention. The infinite lookback provides more flexibility and should improve the modeling of long-distance reorderings and dependencies. The increased computational cost, from linear to quadratic computation, is of little concern as our focus on the simultaneous scenario means that out largest source of latency will be waiting for source context.

Concretely, we maintain a full monotonic attention mechanism and also a soft attention mechanism. Assuming that the monotonic attention component chooses to stop at t_i, MILk first computes soft attention energies

$$u_{i,k} = \text{SoftmaxEnergy}(h_k, s_{i-1}) \tag{12}$$

for $k \in 1, 2, \ldots, t_i$ where $\text{SoftmaxEnergy}(\cdot)$ is an energy function similar to Equation (4). Then, MILk computes a context c_i by

$$c_i = \sum_{j=1}^{t_i} \frac{\exp(u_{i,j})}{\sum_{l=1}^{t_i} \exp(u_{i,l})} h_j \tag{13}$$

Note that a potential issue with this approach is that the model can set the monotonic attention head $t_i = |\mathbf{x}|$ for all i, in which case the approach is equivalent to standard soft attention. We address this issue in the following subsection.

To train models using MILk, we compute the expected value of c_i given the monotonic attention probabilities and soft attention energies. To do so, we must consider every possible path through which the model could assign attention to a given encoder state. Specifically, we can compute the attention distribution induced by MILk by

$$\beta_{i,j} = \sum_{k=j}^{|\mathbf{x}|} \left(\frac{\alpha_{i,k} \exp(u_{i,j})}{\sum_{l=1}^{k} \exp(u_{i,l})} \right) \tag{14}$$

The first summation reflects the fact that h_j can influence c_i as long as $k \geq j$, and the term inside the summation reflects the attention probability associated with some monotonic probability $\alpha_{i,k}$ and the soft attention distribution. This calculation can be computed efficiently using cumulative sum operations by replacing the outer summation with a cumulative sum and the inner operation with a cumulative sum after reversing $\mathbf{u}$. Once we have the $\beta_{i,j}$ distribution, calculating the expected context c_i follows a familiar formula: $c_i = \sum_{j=1}^{|\mathbf{x}|} \beta_{i,j} h_j$.

3.3 Latency-augmented Training

By moving to an infinite lookback, we have gained the full power of a soft attention mechanism over any source tokens that have been revealed up to time t_i. However, while the original monotonic attention encouraged streaming behaviour implicitly due to the restriction on the system's memory, MILk no longer has any incentive to do this. It can simply wait for all source tokens before writing the first target token. We address this problem by training with an objective that interpolates log likelihood with a latency metric.

Sequence-to-sequence models are typically trained to minimize the negative log likelihood, which we can easily augment with a latency cost:

$$L(\theta) = -\sum_{(\mathbf{x},\mathbf{y})} \log p(\mathbf{y}|\mathbf{x};\theta) + \lambda \mathcal{C}(\mathbf{g}) \tag{15}$$

where λ is a user-defined latency weight, $\mathbf{g} = \{g_1, \ldots, g_{|\mathbf{y}|}\}$ is a vector that describes the delay incurred immediately before each target time step (see Section 4.1), and $\mathcal{C}$ is a latency metric that transforms these delays into a cost.

In the case of MILk, g_i is equal to t_i, the position of the monotonic attention head.[1] Recall that during training, we never actually make a hard decision about t_i's location. Instead, we can use $\alpha_{i,j}$,

[1] We introduce g_i to generalize beyond methods with hard attention heads and to unify notation with Ma et al. (2018).

the probability that $t_i = j$, to get expected delay:

$$g_i = \sum_{j=1}^{|\mathbf{x}|} j\alpha_{i,j} \tag{16}$$

So long as our metric is differentiable and well-defined over fractional delays, Equation (15) can be used to guide MILk to low latencies.

3.4 Preserving Monotonic Probability Mass

In the original formulations of monotonic attention (see Section 3.1), it is possible to choose not to stop the monotonic attention head, even at the end of the source sentence. In such cases, the attention returns an all-zero context vector.

In early experiments, we found that this creates an implicit incentive for low latencies: the MILk attention head would stop early to avoid running off the end of the sentence. This implicit incentive grows stronger as our selection probabilities $p_{i,j}$ come closer to being binary decisions. Meanwhile, we found it beneficial to have very-near-to-binary decisions in order to get accurate latency estimates for latency-augmented training. Taken all together, we found that MILk either destabilized, or settled into unhealthily-low-latency regions. We resolve this problem by forcing MILk's monotonic attention head to once stop when it reaches the EOS token, by setting $p_{i,|\mathbf{x}|} = 1$.[2]

4 Measuring Latency

Our plan hinges on having a latency cost that is worth optimizing. To that end, we describe two candidates, and then modify the most promising one to accommodate our training scenario.

4.1 Previous Latency Metrics

Cho and Esipova (2016) introduced Average Proportion (AP), which averages the absolute delay incurred by each target token:

$$\text{AP} = \frac{1}{|\mathbf{x}|\ |\mathbf{y}|} \sum_{i=1}^{|\mathbf{y}|} g_i \tag{17}$$

[2]While training, we perform the equivalent operation of shifting the any residual probability mass from overshooting the source sentence, $1 - \sum_{j=1}^{|\mathbf{x}|} \alpha_{i,j}$, to the final source token at position $|\mathbf{x}|$. This bypasses floating point errors introduced by the parallelized cumulative sum and cumulative product operations (Raffel et al., 2017). This same numerical instability helps explain why the parameterized stopping probability $p_{i,j}$ does not learn to detect the end of the sentence without intervention.

where g_i is delay at time i: the number of source tokens read by the agent before writing the i^{th} target token. This metric has some nice properties, such as being bound between 0 and 1, but it also has some issues. Ma et al. (2018) observe that their wait-k system with a fixed $k = 1$ incurs different AP values as sequence length $|\mathbf{x}| = |\mathbf{y}|$ ranges from 2 (AP $= 0.75$) to ∞ (AP $= 0.5$). Knowing that a very-low-latency wait-1 system incurs at best an AP of 0.5 also implies that much of the metric's dynamic range is wasted; in fact, Alinejad et al. (2018) report that AP is not sufficiently sensitive to detect their improvements to simultaneous MT.

Recently, Ma et al. (2018) introduced Average Lagging (AL), which measures the average rate by which the MT system lags behind an ideal, completely simultaneous translator:

$$\text{AL} = \frac{1}{\tau} \sum_{i=1}^{\tau} g_i - \frac{i-1}{\gamma} \tag{18}$$

where τ is the earliest timestep where the MT system has consumed the entire source sequence:

$$\tau = \text{argmin}_i g_i = |\mathbf{x}| \tag{19}$$

and $\gamma = |\mathbf{y}|/|\mathbf{x}|$ accounts for the source and target having different sequence lengths. This metric has the nice property that when $|\mathbf{x}| = |\mathbf{y}|$, a wait-$k$ system will achieve an AL of k, which makes the metric very interpretable. It also has no issues with sentence length or sensitivity.

4.2 Differentiable Average Lagging

Average Proportion already works as a $\mathcal{C}$ function, but we prefer Average Lagging for the reasons outlined above. Unfortunately, it is not differentiable, nor is it calculable in expectation, due to the argmin in Equation (19). We present Differentiable Average Lagging (DAL), which eliminates the argmin by making AL's treatment of delay internally consistent.

AL's argmin is used to calculate τ, which is used in turn to truncate AL's average at the point where all source tokens have been read. Why is this necessary? We can quickly see τ's purpose by reasoning about a simpler version of AL where $\tau = |\mathbf{y}|$. Table 1 shows the time-indexed lags that are averaged to calculate AL for a wait-3 system. The lags make the problem clear: each position beyond the point where all source tokens have been read ($g_i = |\mathbf{x}|$) has its lag reduced by

Statistics					Scores	
i	1	2	3	4	$\tau = 2$	$\tau = \lvert\mathbf{y}\rvert$
g_i	3	4	4	4		
AL_i	3	3	2	1	AL = 3	AL = 2.25

Table 1: Comparing AL with and without its truncated average, tracking time-indexed lag $\mathrm{AL}_i = g_i - \frac{i-1}{\gamma}$ when $|\mathbf{x}| = |\mathbf{y}| = 4$ for a wait-3 system.

Statistics					Scores
i	1	2	3	4	
g'_i	3	4	5	6	
DAL_i	3	3	3	3	DAL = 3

Table 2: DAL's time-indexed lag $\mathrm{DAL}_i = g'_i - \frac{i-1}{\gamma}$ when $|\mathbf{x}| = |\mathbf{y}| = 4$ for a wait-3 system.

1, pulling the average lag below k. By stopping its average at $\tau = 2$, AL maintains the property that a wait-k system receives an AL of k.

τ is necessary because the only way to incur delay is to read a source token. Once all source tokens have been read, all target tokens appear instantaneously, artificially dragging down the average lag. This is unsatisfying: the system lagged behind the source speaker while they were speaking. It should continue to do so after they finished.

AL solves this issue by truncating its average, enforcing an implicit and poorly defined delay for the excluded, problematic tokens. We propose instead to enforce a minimum delay for writing any target token. Specifically, we model each target token as taking at least $\frac{1}{\gamma}$ units of time to write, mirroring the speed of the ideal simultaneous translator in AL's Equation (18). We wrap $\mathbf{g}$ in a $\mathbf{g}'$ that enforces our minimum delay:

$$g'_i = \begin{cases} g_i & i = 1 \\ \max\left(g_i, g'_{i-1} + \frac{1}{\gamma}\right) & i > 1 \end{cases} \tag{20}$$

Like g_i, g'_i represents the amount of delay incurred just before writing the i^{th} target token. Intuitively, the $\max$ enforces our minimum delay: g'_i is either equal to g_i, the number of source tokens read, or to $g'_{i-1} + \frac{1}{\gamma}$, the delay incurred just before the previous token, plus the time spent writing that token. The recurrence ensures that we never lose track of earlier delays. With $\mathbf{g}'$ in place, we can define our Differentiable Average Lagging:

$$\mathrm{DAL} = \frac{1}{|\mathbf{y}|} \sum_{i=1}^{|\mathbf{y}|} g'_i - \frac{i-1}{\gamma} \tag{21}$$

DAL is equal to AL in many cases, in particular, when measuring wait-k systems for sentences of equal length, both always return a lag of k. See Table 2 for its treatment of our wait-3 example. Having eliminated τ, DAL is both differentiable and calcuable in expectation. Cherry and Foster (2019) provide further motivation and analysis for DAL, alongside several examples of cases where DAL yields more intuitive results than AL.

5 Experiments

We run our experiments on the standard WMT14 English-to-French (EnFr; 36.3M sentences) and WMT15 German-to-English (DeEn; 4.5M sentences) tasks. For EnFr we use a combination of newstest 2012 and newstest 2013 for development and report results on newstest 2014. For DeEn we validate on newstest 2013 and then report results on newstest 2015. Translation quality is measured using detokenized, cased BLEU (Papineni et al., 2002). For each data set, we use BPE (Sennrich et al., 2016) on the training data to construct a 32,000-type vocabulary that is shared between the source and target languages.

5.1 Model

Our model closely follows the RNMT+ architecture described by Chen et al. (2018) with modifications to support streaming translation. It consists of a 6 layer LSTM encoder and an 8 layer LSTM decoder with additive attention (Bahdanau et al., 2014). All streaming models including wait-k, MoChA and MILk use unidirectional encoders, while offline translation models use a bidirectional encoder. Both encoder and decoder LSTMs have 512 hidden units, per gate layer normalization (Ba et al., 2016), and residual skip connections after the second layer. The models are regularized using dropout with probability 0.2 and label smoothing with an uncertainty of 0.1 (Szegedy et al., 2016). Models are optimized until convergence using data parallelism over 32 P100s, using Adam (Kingma and Ba, 2015) with the learning rate schedule described in Chen et al. (2018) and a batch size of 4,096 sentence-pairs per GPU. Checkpoints are selected based on development loss. All streaming models use greedy decoding, while offline models use beam search with a beam size of 20.

We implement soft attention, monotonic attention, MoChA, MILk and wait-k as instantiations

	unpreserved		preserved	
λ	BLEU	DAL	BLEU	DAL
0.0	27.7	21.0	27.7	27.9
0.1	27.0	13.6	27.6	10.5
0.2	25.7	11.6	27.5	8.7

Table 3: Varying MILk's λ with and without mass preservation on the DeEn development set.

n	BLEU	DAL
0	3.4	24.2
1	10.8	12.9
2	24.6	12.3
3	27.5	10.4
4	**27.5**	**8.7**
6	26.3	7.2

Table 4: Varying MILk's discreteness parameter n with λ fixed at 0.2 on the DeEn development set.

of an attention interface in a common code base, allowing us to isolate their contributions. By analyzing development sentence lengths, we determined that wait-k should employ a emission rate of 1 for DeEn, and 1.1 for EnFr.

5.2 Development

We tuned MILk on our DeEn development set. Two factors were crucial for good performance: the preservation of monotonic mass (Section 3.4), and the proper tuning of the noise parameter n in Equation 11, which controls the discreteness of monotonic attention probabilities during training.

Table 3 contrasts MILk's best configuration before mass preservation against our final system. Before preservation, MILk with a latency weight $\lambda = 0$ still showed a substantial reduction in latency from the maximum value of 27.9, indicating an intrinsic latency incentive. Furthermore, training quickly destabilized, resulting in very poor trade-offs for λs as low as 0.2.

After modifying MILk to preserve mass, we then optimized noise with λ fixed at a low but relevant value of 0.2, as shown in Table 4. We then proceeded the deploy the selected value of $n = 4$ for testing both DeEn and EnFr.

5.3 Comparison with the state-of-the-art

We compare MILk to wait-k, the current state-of-the-art in simultaneous NMT. We also include MILk's predecessors, Monotonic Attention and MoChA, which have not previously been evaluated with latency metrics. We plot latency-quality curves for each system, reporting quality using BLEU, and latency using Differentiable Average Lagging (DAL), Average Lagging (AL) or Average Proportion (AP) (see Section 4). We focus our analysis on DAL unless stated otherwise. MILk curves are produced by varying the latency loss weight λ,[3] wait-k curves by varying k,[4] and MoChA curves by varying chunk size.[5] Both MILk and wait-k have settings ($\lambda = 0$ and $k = 300$) corresponding to full attention.

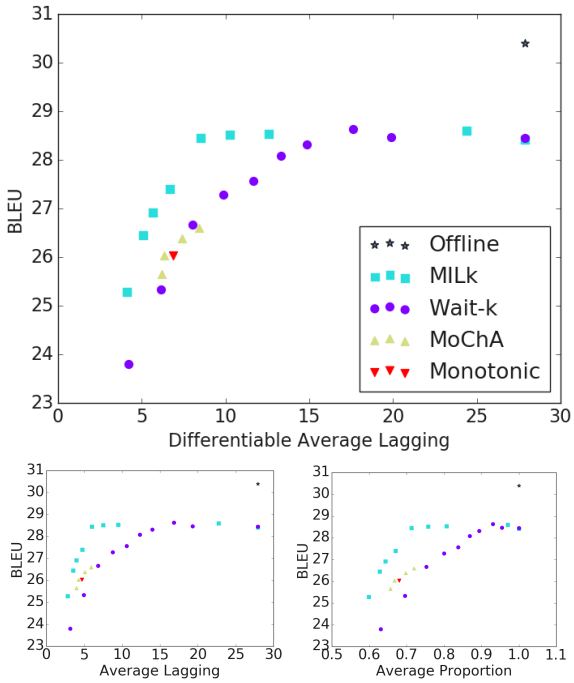


Figure 2: Quality-latency comparison for German-to-English WMT15 (DeEn) with DAL (upper), AL (lower-left), AP (lower-right).

Results are shown in Figures 2 and 3.[6] For DeEn, we begin by noting that MILk has a clear separation above its predecessors MoChA and Monotonic Attention, indicating that the infinite lookback is indeed a better fit for translation. Furthermore, MILk is consistently above wait-k for lags between 4 and 14 tokens. MILk is able to retain the quality of full attention (28.4 BLEU) up to a lag of 8.5 tokens, while wait-k begins to fall off for lags below 13.3 tokens. At the lowest comparable latency (4 tokens), MILk is 1.5 BLEU points

[3] $\lambda = 0.75, 0.5, 0.4, 0.3, 0.2, 0.1, 0.05, 0.01, 0.0$

[4] $k = 2, 4, 6, 8, 10, 12, 14, 16, 20, 24, 300$

[5] $cs = 1$ (Monotonic Attention), 2, 4, 8, and 16

[6] Full sized graphs for all latency metrics, along with the corresponding numeric scores are available in Appendix A, included as supplementary material.

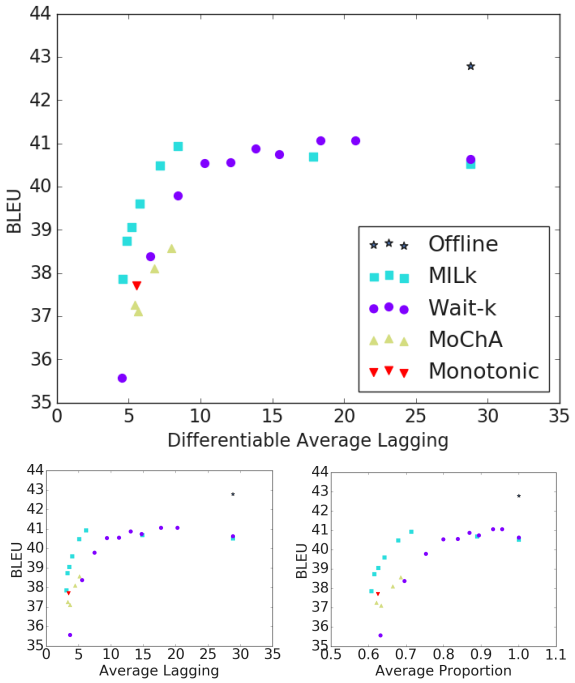


Figure 3: Quality-latency comparison for English-to-French WMT14 (EnFr) with DAL (upper), AL (lower-left), AP (lower-right).

ahead of wait-k.

EnFr is a much easier language pair: both MILk and wait-k maintain the BLEU of full attention at lags of 10 tokens. However, we were surprised to see that this does not mean we can safely deploy very low ks for wait-k; its quality drops off surprisingly quickly at $k = 8$ (DAL=8.4, BLEU=39.8). MILk extends the flat "safe" region of the curve out to a lag of 7.2 (BLEU=40.5). At the lowest comparable lag (4.5 tokens), MILk once again surpasses wait-k, this time by 2.3 BLEU points.

The $k = 2$ point for wait-k has been omitted from all graphs to improve clarity. The omitted BLEU/DAL pairs are 19.5/2.5 for DeEn and 28.9/2.9 for EnFr, both of which trade very large losses in BLEU for small gains in lag. However, wait-k's ability to function at all at such low latencies is notable. The configuration of MILk tested here was unable to drop below lags of 4.

Despite MILk having been optimized for DAL, MILk's separation above wait-k only grows as we move to the more established metrics AL and AP. DAL's minimum delay for each target token makes it far more conservative than AL or AP. Unlike DAL, these metrics reward MILk and its predecessors for their tendency to make many consecutive writes in the middle of a sentence.

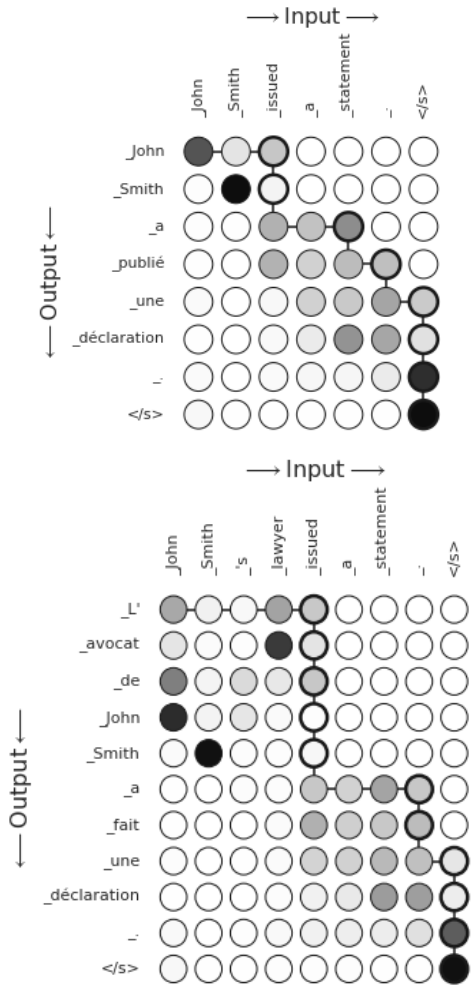


Figure 4: Two EnFr sentences constructed to contrast MILk's handling of a short noun phrase *John Smith* against the longer *John Smith's lawyer*. Translated by MILk with $\lambda = 0.2$.

5.4 Characterizing MILK's schedule

We begin with a qualitative characterization of MILk's behavior by providing diagrams of MILk's attention distributions. The shade of each circle indicates the strength of the soft alignment, while bold outlines indicate the location of the hard attention head, whose movement is tracked by connecting lines.

In general, the attention head seems to loosely follow noun- and verb-phrase boundaries, reading one or two tokens past the end of the phrase to ensure it is complete. This behavior and its benefits are shown in Figure 4, which contrast the simple noun phrase *John Smith* against the more complex *John Smith's laywer*. By waiting until the end of both phrases, MILk is able to correctly re-order *avocat* (*lawyer*).

Figure 5 shows a more complex sentence drawn

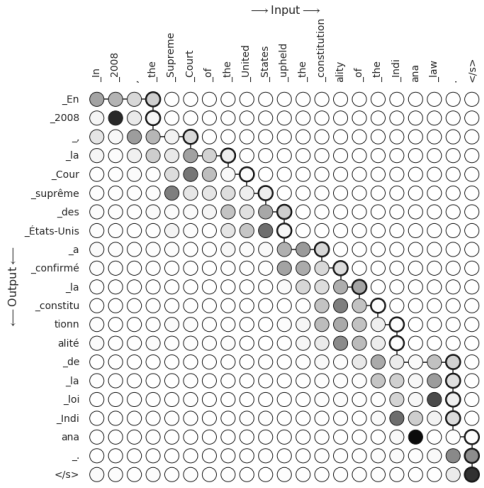


Figure 5: An example EnFr sentence drawn from our development set, as translated by MILk with $\lambda = 0.2$.

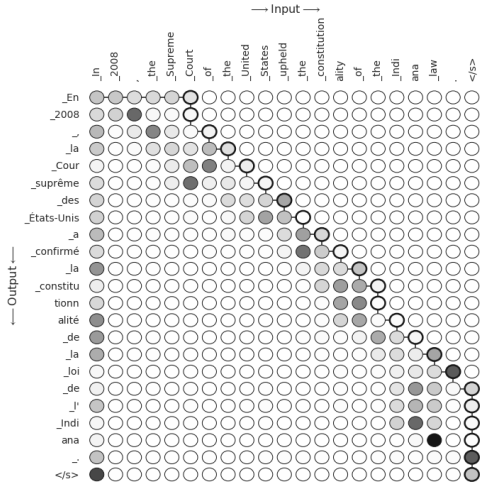


Figure 6: An example EnFr sentence drawn from our development set, as translated by wait-6.

from our development set. MILk gets going after reading just 4 tokens, writing the relatively safe, *En 2008*. It does wait, but it saves its pauses for tokens with likely future dependencies. A particularly interesting pause occurs before the *de* in *de la loi*. This preposition could be either *de la* or *du*, depending on the phrase it modifies. We can see MILk pause long enough to read one token after *law*, allowing it to correctly choose *de la* to match the feminine *loi* (*law*).

Looking at the corresponding wait-6 run in Figure 6, we can see that wait-6's fixed schedule does not read *law* before writing the same *de*. To its credit, wait-6 anticipates correctly, also choosing *de la*, likely due to the legal context provided by the nearby phrase, *the constitutionality*.

We can also perform a quantitative analysis of

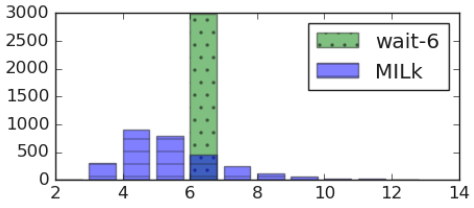


Figure 7: Histogram of initial delays for MILk ($\lambda = 0.2$) and wait-6 on the EnFr development set.

MILk's adaptivity by monitoring its initial delays; that is, how many source tokens does it read before writing its first target token? We decode our EnFr development set with MILk $\lambda = 0.2$ as well as wait-6 and count the initial delays for each.[7] The resulting histogram is shown in Figure 7. We can see that MILk has a lot of variance in its initial delays, especially when compared to the near-static wait-6. This is despite them having very similar DALs: 5.8 for MILk and 6.5 for wait-6.

6 Conclusion

We have presented Monotonic Infinite Lookback (MILk) attention, an attention mechanism that uses a hard, monotonic head to manage the reading of the source, and a soft traditional head to attend over whatever has been read. This allowed us to build a simultaneous NMT system that is trained jointly with its adaptive schedule. Along the way, we contributed latency-augmented training and a differentiable latency metric. We have shown MILk to have favorable quality-latency trade-offs compared to both wait-k and to earlier monotonic attention mechanisms. It is particularly useful for extending the length of the region on the latency curve where we do not yet incur a major reduction in BLEU.

References

Ashkan Alinejad, Maryam Siahbani, and Anoop Sarkar. 2018. Prediction improves simultaneous neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3022–3027. Association for Computational Linguistics.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. *arXiv e-prints*, page arXiv:1607.06450.

[7] Wait-6 will have delays different from 6 only for source sentences with fewer than 6 tokens.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Srinivas Bangalore, Vivek Kumar Rangarajan Sridhar, Prakash Kolan, Ladan Golipour, and Aura Jimenez. 2012. Real-time incremental speech-to-speech translation of dialogs. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 437–445. Association for Computational Linguistics.

Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Mike Schuster, Noam Shazeer, Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. 2018. The best of both worlds: Combining recent advances in neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 76–86. Association for Computational Linguistics.

Colin Cherry and George Foster. 2019. Thinking Slow about Latency Evaluation for Simultaneous Machine Translation. *arXiv e-prints*, page arXiv:1906.00048.

Chung-Cheng Chiu and Colin Raffel. 2018. Monotonic chunkwise attention. In *International Conference on Learning Representations*.

Kyunghyun Cho and Masha Esipova. 2016. Can neural machine translation do simultaneous translation? *CoRR*, abs/1606.02012.

Fahim Dalvi, Nadir Durrani, Hassan Sajjad, and Stephan Vogel. 2018. Incremental decoding and training methods for simultaneous translation in neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 493–499. Association for Computational Linguistics.

Christian Fügen, Alex Waibel, and Muntsin Kolss. 2007. Simultaneous translation of lectures and speeches. *Machine Translation*, 21(4):209–252.

Alvin Grissom II, He He, Jordan Boyd-Graber, John Morgan, and Hal Daumé III. 2014. Don't until the final verb wait: Reinforcement learning for simultaneous machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1342–1352. Association for Computational Linguistics.

Jiatao Gu, Graham Neubig, Kyunghyun Cho, and Victor O.K. Li. 2017. Learning to translate in real-time with neural machine translation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1053–1062. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Mingbo Ma, Liang Huang, Hao Xiong, Kaibo Liu, Chuanqiang Zhang, Zhongjun He, Hairong Liu, Xing Li, and Haifeng Wang. 2018. STACL: Simultaneous Translation with Integrated Anticipation and Controllable Latency. *arXiv e-prints*, page arXiv:1810.08398.

Yusuke Oda, Graham Neubig, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2014. Optimizing segmentation strategies for simultaneous speech translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 551–556. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*.

Ofir Press and Noah A. Smith. 2018. You May Not Need Attention. *arXiv e-prints*, page arXiv:1810.13409.

Colin Raffel, Minh-Thang Luong, Peter J. Liu, Ron J. Weiss, and Douglas Eck. 2017. Online and linear-time attention by enforcing monotonic alignments. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2837–2846, International Convention Centre, Sydney, Australia. PMLR.

Vivek Kumar Rangarajan Sridhar, John Chen, Srinivas Bangalore, Andrej Ljolje, and Rathinavelu Chengalvarayan. 2013. Segmentation strategies for streaming speech translation. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 230–238. Association for Computational Linguistics.

Harsh Satija and Joelle Pineau. 2016. Simultaneous machine translation using deep reinforcement learning. In *Proceedings of the Abstraction in Reinforcement Learning Workshop*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725. Association for Computational Linguistics.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826.

Supplementary Material

We have provided a separate file containing supplementary material. Its Appendix A contains full-sized graphs and numeric scores to support our primary experimental comparison in Section 5.3.